TP Cryptographie
Méthode du sac-à-dos

Partie 1 - Préparation

NOMS et Prénoms des étudiants :
Groupe :

1 - Les boucles "for"

```
> restart;
```

La syntaxe n'est pas la même qu'en langage C, mais la signification est facile à comprendre :

```
> u:=0;
  for i from 1 to 10
  do
    u:=u+i;
  end do;
  u;
```

Vérification :

```
> (10+1)*10/2;
```

Le mot-clé "by" permet de changer le pas de l'itération :

```
> u;
  for j from 10 to 1 by -1
  do
    u:=u-j;
  end do;
  u;
```

2 - Les listes de listes

Comme en langage C, les éléments d'une liste peuvent être eux-mêmes des listes.

```
> listedouble:=[[6,7],[18,20,4,2],[5,17,16,4,10,15]];
```

Un élément de la liste "listedouble" est une liste :

```
> listedouble[2];
```

Un élément d'un élément de la liste "listedouble" est un nombre :

```
> listedouble[3][6];
```

On écrit chacun des éléments de la liste :

```
> for j from 1 to 3
  do
    print(listedouble[j]);
  end do;
```

On écrit chacun des nombres contenus dans les éléments de la liste :

```
> for j from 1 to 3
  do
    print("liste numéro"=j);
    for i from 1 to 2*j
    do
      print(listedouble[j][i]);
    end do;
  end do;
```

3 - Exercice : Un petit sac-à-dos

Vous avez vu toutes les commandes nécessaires pour traiter un petit exercice sur la méthode du sac-à-dos.

```
> restart;
```

Etape 1 : Création du code secret

On choisit la longueur n du sac-à-dos.

```
> n:=5;
```

Question 1 : Quelle est l'influence du choix de n sur les mots binaires que le code permet de chiffrer ?

On choisit le sac-à-dos facile SADF. On le range dans une liste appelée "SADF".

```
> SADF:=[3,5,10,21,42];
```

Question 2 : Vérifier que SADF est bien un sac-à-dos facile.

On choisit le module N.

```
> N:=155;
```

Question 3 : Vérifier que N est bien un module acceptable en utilisant la commande "add" (voir TP1).

On choisit le compliqueur e.

```
> e:=13;
```

Question 4 : Vérifier que e est bien un compliqueur acceptable.

Question 5 : Calculer le sac-à-dos difficile SADD et ranger la liste des cinq nombres obtenus dans la variable "SADD" en utilisant la commande "seq" (voir TP1) .

Question 6 : Calculer le faciliteur et ranger le résultat dans la variable "d".

Question 7 : Vérifier que d est bien le faciliteur.

Etape 2 : Chiffrer

On donne ci-dessous le message binaire m que vous devez chiffrer :

```
> m:=[[0,1,0,1,1], [0,1,0,1,0], [0,1,1,1,0]];
```

Question 8 : Pourquoi m est-il découpé en mots de 5 bits ?

Question 9 : Chiffrer le message binaire m en utilisant les commande "seq" et "add", et ranger la liste des trois nombres obtenus dans la variable "crypto". Le bit de poids faible de chaque mot binaire m[j] est le bit d'indice 1 (c'est-à-dire m[j][1]) et correspond à l'élément d'indice 1 du sac-à-dos (c'est-à-dire SADD[1]). On vous demande simplement de remplacer les "..." par ce qui convient.

```
> crypto:=[seq(add(m[j][i]*...,i=1..n) mod N,j=1.. ...)];
```

Etape 3 : Déchiffrer

La première étape du déchiffage consiste à transformer le message représenté à l'aide de SADD en un message représenté à l'aide de SADF.

Question 10 : Effectuer cette transformation sur le cryptogramme "crypto" et ranger la liste des trois nombres obtenus dans la variable "decrypto" en utilisant la commande "seq".

Question 11 : Terminer le déchiffrage (obtenir le message binaire). Pour cela, on commence par créer un message binaire vide appelé "r" constitué de 3 mots de 5 bits, puis on utilise l'algorithme glouton pour calculer chacun des bits de r. On vous demande simplement de remplacer les "..." par ce qui convient.

```
> r:=[seq([seq(0,i=1..n)],j=1..3)]:
> for j from 1 to 3
  do
    dec:=decrypto[j];
    for i from n to 1 by -1
    do
      if dec>=SADF[i]
         then
            r[j][i]:=...;
            dec:=dec-SADF[i];
         else
            r[j][i]:=...;
      end if;
    end do;
  end do:
> r;
```

Question 12 : Comment pouvez-vous vérifier que votre résultat est correct ?

À partir de maintenant, vous allez faire de la cryptographie en (presque) vraie grandeur dans les parties 2,3 et 4. Dans la partie 2, vous allez créer votre code secret. Dans la partie 3, vous allez chiffrer un message secret et l'envoyer. Enfin, dans la partie 4, vous allez recevoir un message secret et le déchiffrer.
Il est bien sûr recommandé de vous aider de l'exercice sur le petit sac-à-dos que vous venez de traiter lorsque c'est nécessaire.

Partie 2 - Créer un code secret et publier la clé publique

```
> restart;
```

Le but de cette partie est de créer un code secret "sac-à-dos" de longueur comprise entre 30 et 60 et composé de nombres d'au moins 12 chiffres décimaux.

Pour commencer, vous devez fixer la longueur de votre sac-à-dos. Choisissez un nombre entre 30 et 60 et ranger-le ci-dessous dans longsad :

```
> longsad:=...;
```

Pour assurer que vous ayez tous des codes différents, vous allez créer votre code en utilisant des nombres choisis "au hasard". (cf TP2 RSA)

```
> Seed:=...;
```

Les commandes suivantes permettent de créer un sac-à-dos facile de longsad nombres à partir de nombres "au hasard" :

```
> SADF:=[seq(0,i=1..longsad)]:
  for i from 1 to longsad
  do
     SADF[i]:=rand()+add(SADF[j],j=1..i)
  end do:
  SADF;
```

Question 1 : Expliquer pourquoi les nombres de la liste SADF constituent obligatoirement un sac-à-dos facile.

Réponse :

Pour éviter les accidents de manipulation, coller votre SADF ci-dessous :

```
> SADF:=...;
```

Question 2 : Appliquer la même méthode qu'à la question 1 pour créer un module acceptable, et ranger le résultat dans la variable "N".

Question 3 : On tente de créer un compliqueur par la commande ci-dessous. Vérifier si e est un compliqueur acceptable. Si ce n'est pas le cas, recommencer jusqu'à obtenir un compliqueur acceptable.

```
> e:=N-rand();
```

Comme précédemment coller N et e ci-dessous :

```
> N:=...;
  e:=...;
```

Question 4 : Calculer le sac-à-dos difficile et ranger le résultat dans la variable "SADD".

Question 5 : Calculer le faciliteur correspondant au compliqueur e et ranger le résultat dans la variable "d".

Question 6 : Il ne vous reste plus qu'à publier la partie publique de votre code secret. Pour cela, vous devez aller sur le forum de cryptographie de votre groupe de TP, sur le serveur web de l'IUT en ligne, auquel vous avez accès pendant cette séance. Vous devez créer sur ce forum votre boîte aux lettres, c'est-à-dire un nouveau sujet intitulé "Boîte aux lettres SAD de (*vos deux noms*)".
Le premier message de ce sujet sera en fait la partie publique de votre code secret (encore appelée "clé publique"). Attention toutefois à ne poster aucun élément de la partie privée de votre code !
Comme votre sac-à-dos est probablement trop long pour faire un Copier/Coller, vous pouvez utiliser le Bloc-Note de Windows.

Indiquez ci-après de quoi est constituée cette "clé publique".
Réponse :

Partie 3 - Envoyer un message secret

Vous devez envoyer un message secret à vos voisins de droite.
Écrire ici les noms des destinataires :

Votre message doit comporter au moins 3 lignes de texte.

```
> texte:="Tapez votre texte entre les guillemets.";
```

Pour convertir votre message texte en message binaire, c'est-à-dire en une liste de mots binaires, on vous donne la fonction "encodebin" ci-dessous (voir TP1).
Ne pas oublier d'appuyer d'abord sur "Entrée" pour enregistrer le code dans la commande "encodebin".

```
> encodebin:=proc(lb,t)
  local nbcar,listeascii, listenum,nb,bin,nbbit,nbbloc,fin,i;
  nbcar:=length(t);
  listeascii:=convert(t,bytes);
  listenum:=[seq(listeascii[i]*10^(3*(i-1)),i=1..nbcar)];
  nb:=add(listenum[i],i=1..nbcar);
  bin:=convert(nb,base,2);
  nbbit:=nops(bin);
  nbbloc:=ceil(nbbit/lb);
  fin:=nbbloc*lb-nbbit;
  return([seq([seq(bin[i],i=1+(j-1)*lb..j*lb)],j=1..nbbloc-1),
  [seq(bin[i],i=nbbit-(lb-fin)+1..nbbit),seq(0,i=1..fin)]]);
  end proc:
```

La fonction encodebin prend en entrée la longueur des blocs "lb" et le texte "t" (c'est votre message secret), et renvoie la liste des mots binaires de lb bits obtenus à partir des codes ASCII des caractères du texte.

Question 1 : Coller ci-dessous la clé publique de vos correspondants qui se trouve sur le forum, et la ranger dans les variables "SADDbis" et "Nbis".

Les commandes suivantes calculent les longueurs des sacs-à-dos de vos correspondants :

```
> longsadbis:=nops(SADDbis);
```

Question 2 : Convertir votre message secret texte en un message secret binaire en utilisant la fonction "encodebin". Ranger le résultat dans la variable "messbin".

```
> messbin:=...;
```

La commande ci-dessous compte le nombre de blocs de votre message binaire m.

```
> nbrebloc:=nops(messbin);
```

Question 3 : Chiffrez votre message binaire ci-dessous et ranger le résultat dans la variable "crypto".

Question 4 : Poster votre cryptogramme sur le forum dans la boîte aux lettres SAD de vos destinataires (c'est-à-dire "Répondre" à leur premier message).

Partie 4 - Recevoir un message secret

Vous devez recevoir un message de vos voisins de gauche.
Écrire ici les noms des expéditeurs :

Question 1 : Récupérer sur le forum le message secret qui vous est destiné, et le ranger dans la variable "cryptobis".

```
> cryptobis :=...;
```

Question 2 : Calculer le nombre de blocs du message qui vous est destiné, et ranger le résultat dans la variable "nbreblocbis".

Question 3 : Déchiffrer le message qui vous a été envoyé et ranger le résultat dans la variable "decrypto".

Question 4 : Calculer le message binaire et ranger le résultat dans la variable "r".

Question 5 : Il ne vous reste plus qu'à convertir en texte le message binaire que vous avez déchiffré. Pour cela, on vous donne la fonction "decodebin" ci-dessous :

```
> decodebin:=proc(m)
  local nbblocbis,lbis,binbis,listedec,nbchbis,nbcarbis,
  listedecbis,listeasciibis,i;
  nbblocbis:=nops(m):
  lbis:=nops(m[1]):
  binbis:=[seq(seq(m[i][j],j=1..lbis),i=1..nbblocbis)]:
  listedec:=convert(binbis,base,2,10):
  nbchbis:=nops(listedec):
  nbcarbis:=ceil(nbchbis/3):
  listedecbis:=[seq(listedec[i],i=1..nbchbis),seq(0,j=1..3*
  nbcarbis-nbchbis)];
  listeasciibis:=[seq(listedecbis[3*(i-1)+1]+listedecbis[3*(i-1)
  +2]*10+listedecbis[3*(i-1)+3]*100,i=1..nbcarbis)]:
  return(convert(listeasciibis,bytes));
  end proc:
```

La fonction decodebin prend en entrée un message binaire r et renvoie le texte qu'il représente (cf TP1).

Partie facultative - Décrypter un code "sac-à-dos"

Un espion a évidemment accès à la clé publique (à savoir le sac-à-dos difficile SADD et le module M). S'il intercepte un cryptogramme, son travail consiste à essayer de retrouver le message binaire dont il est issu. Une méthode brutale consiste à essayer de chiffrer tous les messages binaires possibles, et de voir lequel donne le même cryptogramme que celui qui a été intercepté. Cette méthode est très lente, et même impraticable pour des sac-à-dos un peu longs (voir les tests ci-dessous). Cependant,

on a longtemps cru qu'il ne pouvait exister aucune méthode plus efficace pour les sac-à-dos difficiles (en termes techniques, on dit que "le problème du sac-à-dos est NP-complet"). Puis on a finalement découvert un algorithme assez compliqué mais très astucieux (appelé "LLL") permettant de retrouver rapidement le message binaire de départ pour certains types de sac-à-dos difficiles, parmi lesquels ceux qu'on fabrique à partir d'un sac-à-dos facile en utilisant le calcul modulaire, c'est-à-dire ceux qui servent pour la cryptographie. C'est pour cette raison que la méthode du sac-à-dos n'est plus utilisée actuellement, contrairement à la méthode RSA.

On donne ci-dessous la fonction "decrypte", qui prend en entrée un cryptogramme "m", le sac-à-dos difficile "S" et le module "modu" qui ont servi à fabriquer m, et qui renvoie le texte de départ, calculé par la méthode brutale.

```
> decrypte:=proc(m,S,modu)
  local l,nbbl,m1,m2,c,i,j,k,m3;
  l:=nops(S):
  nbbl:=nops(m):
  m1:=[seq(0,k=1..nbbl)]:
  for k from 1 to nbbl do
     j:=2^l;
     c:=0;
     m2:=convert(j,base,2);
     while c<>m[k] do
       m2:=convert(j,base,2);
       c:=0;
       for i from 1 to l do
          c:=c+S[i]*m2[i] mod modu;
       end do;
       j:=j+1;
     end do;
     m1[k]:=m2;
  end do:
  m3:=[seq([seq(m1[i][j],j=1...l)],i=1..nbbl)]:
  return(decodebin(m3));
  end proc:
```

ATTENTION : Sauvegarder votre fichier avant de lancer ces calculs, pour éviter les accidents !!!!

Vous pouvez déterminer ci-dessous quelques temps de calcul (donnés en secondes) :

Sac-à-dos de 5 nombres :

```
> crypto:=[12958403401899, 12811930407095, 4695463330376,
  11086701239444, 2375441627213, 11086701239444, 0,
  12902983477849, 10940228244640, 10582961774686, 2375441627213,
  12902983477849, 8620206541477, 2375441627213, 2242567722178,
  2228968632409, 4191723865618, 2242567722178, 12902983477849,
  279812488969, 2242567722178, 10582961774686, 10940228244640,
```

```
   12958403401899, 8620206541477, 11142121163494, 12811930407095,
   2228968632409, 10940228244640, 2320021703163]:
   SADD:=[8766679536281, 2320021703163, 2375441627213,
   12889384388080, 1816282238405]:
   N:=13035857382884:
   debut:=time():
   decrypte(crypto,SADD,N);
   time()-debut;
```

Sac-à-dos de 10 nombres :

```
> crypto:=[252460691533709, 268455331677469, 168953064937190,
   30550051428009, 272070706924870, 108690246996356,
   153339860372930, 196483956213482, 180439269345618,
   209156630013978, 226636681560644, 408321129173715,
   369081680374270, 156054510506193, 402224526677205]:
   SADD:=[368578389536433, 99502266740279, 78140195568347,
   59072821032990, 346060372545755, 215741820001340,
   285632821812841, 136250422248845, 399637626909837,
   350298954504504]:
   N:=410561772445338:
   debut:=time():
   decrypte(crypto,SADD,N);
   time()-debut;
```

Sac-à-dos de 15 nombres :

```
> crypto:=[6810767122150873, 1748115531447645, 11949927928860411,
   5685056522210311, 9006091730159948, 1702160349221874,
   5344473331076554, 4496236476801280, 8024182263337099,
   2550493836398540]:
   SADD:=[3996802190841387, 5458363073471026, 5739870372837946,
   2958086269945437, 5098609883374081, 3450775497028333,
   1709059595628920, 3417447818667574, 9860892456282777,
   4093616754526914, 10043692803792531, 5113042279598221,
   5461558821378281, 680357315747039, 4296694180859611]:
   N:=13140616317461547:
   debut:=time():
   decrypte(crypto,SADD,N);
   time()-debut;
```

Sac-à-dos de 16 nombres :

```
> crypto:=[17175589667255296, 20552103342833480,
   22885576991847199, 26276019539777267, 17454365184365829,
   7849786504425110, 15082357244407043, 24682084960812999,
   23498766738391999, 14767789208217319]:
   SADD:=[1005529519996366, 2132139789303786, 14767789208217319,
   19270215672522964, 580193839907221, 11032650773375324,
   2771897029969240, 15139976379874918, 24036897255887119,
```

```
   18787910528286656, 24267851321128143, 15364445832879149,
   13584714780538357, 8549192652129947, 2261590042494273,
   7999821774480567]:
   N:=26281085056546925:
   debut:=time():
   decrypte(crypto,SADD,N);
   time()-debut;
```

Sac-à-dos de 17 nombres :

```
> crypto:=[34496539445761027, 28121616844575588,
   50460479836068438, 32140910806841731, 26329012795355422,
   7017351852017840, 2981207196066067 5, 9495142939932631,
   27093345458174897]:
   SADD:=[15540579608626787, 44682395677341462, 41862676169713233,
   452462906970853, 31453941831599097, 44227219806863478,
   33046812202011885, 19650169184471416, 34325978655839073,
   2130229321528432, 2902375607152396, 5001783490284703,
   559711487604899, 50988998639372349, 39719080730314236,
   17544626802820014, 32446866845031475]:
   N:=52561810939655395:
   debut:=time():
   decrypte(crypto,SADD,N);
   time()-debut;
```

Sac-à-dos de 18 nombres :

```
> crypto:=[87497699387631383, 20954758673000136,
   44982597170823017, 34014540637117252, 84348342007779938,
   85357569994574891, 92425388762263271, 29573658573782031,
   79382779524412037]:
   SADD:=[64081240633539229, 17399582026679327, 79382779524412037,
   102392231011670012, 56260705515390673, 18307519935479848,
   9249324310337987, 96860720447067225, 85733426152756759,
   67762344909401454, 70866493900619170, 80285717543989972,
   51439053761786306, 53552988962144943, 53749689213996692,
   6743157371364883, 6260094410876177, 97471343048845387]:
   N:=105123812163686636:
   debut:=time():
   decrypte(crypto,SADD,N);
   time()-debut;
```

On constate expérimentalement que chaque fois qu'on ajoute un nombre au sac-à-dos, le temps de décryptage est multiplié par environ 2.
Par exemple, avec un sac-à-dos de 22 nombres, ce temps est de 45 minutes. Pour les sac-à-dos d'au moins 30 nombres, comme ceux que vous utilisez dans ce TP, le temps de décryptage d'un message dépasse la semaine. On peut donc considérer que de tels codes sont relativement sûrs à votre échelle.

Voici quelques temps expérimentaux supplémentaires :

Sac-à-dos de 19 nombres : 439.942 secondes soit environ 7 min
Sac-à-dos de 20 nombres : 792.412 secondes soit environ 13 min
Sac-à-dos de 21 nombres : 1727.143 secondes soit environ 29 min
Sac-à-dos de 22 nombres : 2770.312 secondes soit environ 46 min